# COURS SYSTÈME D'EXPLOITATION

# Gestion de mémoire

# Plan

- **Introduction**
  - Définition de la mémoire principale
  - Gestionnaire de mémoire : Objectifs et tâches à remplir
- **Adresse physique/ Adresse logique**
  - Définition
  - MMU
  - Registres base et limite
- **Présentation de la mémoire dans un système**
  - Monoprogrammé (mono-processus)
  - Multiprogrammé (multiprocessus)
- **Techniques de gestion de la mémoire**
  - Swapping
  - Partitionnement fixes sans va-et-vient
  - Partitionnement variables et va-et-vient

# Mémoire: Définition

- La mémoire principale est l’unité où se sont stockés les espaces d’adressage des processus en cours d’exécution.
- Une mémoire se présente comme un ensemble de cellules ou d'unités de rangement identiques destinées à mémoriser de l'information.
- Chaque unité de rangement est identifiée par un numéro qu'on appelle Adresse.

# Gestionnaire de mémoire: objectifs

La gestion de la mémoire a deux objectifs majeurs :

- Le partage de la mémoire physique entre les programmes et les données des processus prêts
  - Allouer un espace d'adressage indépendant à chaque processus.
  - Le partage doit être transparent : l'utilisateur ne doit pas savoir si la mémoire est partagée ou pas (illusion d'une mémoire infinie).
- La protection des espaces d'adressages des processus.

# Gestionnaire de mémoire: tâches

Le gestionnaire de la mémoire doit remplir principalement les tâches suivantes :

- Connaitre l'état de la mémoire (les parties libres et occupées de la mémoire).
- Allouer un espace mémoire à un processus avant son exécution.
- Récupérer l'espace alloué à un processus lorsque celui-ci se termine.
- Traiter le va-et-vient (Swapping) entre le disque et la mémoire principale lorsque cette dernière ne peut pas contenir tous les processus.
- Posséder un mécanisme de conversion des adresses physiques en adresses **relatives ou virtuelles**.

# Plan

- **Introduction**
- **Adresse physique/ Adresse logique**
  - Définition
  - MMU
  - Registres base et limite
- **Présentation de la mémoire dans un système**
- **Techniques de gestion de la mémoire**

# Adresses logique / physique

- Une adresse logique est une adresse qui se réfère à une position par rapport au programme lui-même. Elle est indépendante de la position du programme en mémoire physique.
- Une adresse physique est une adresse vue par la mémoire principale.
  - C'est elle qui est chargée dans le registre d'adresse de la mémoire.

# Adresses logique / physique

Remarques :

- Mémoire physique : la mémoire principale RAM de la machine.
  - Adresses physiques: les adresses de cette mémoire.
- Mémoire logique: l'espace d'adressage d'un processus.
  - Adresses logiques: les adresses dans cet espace.

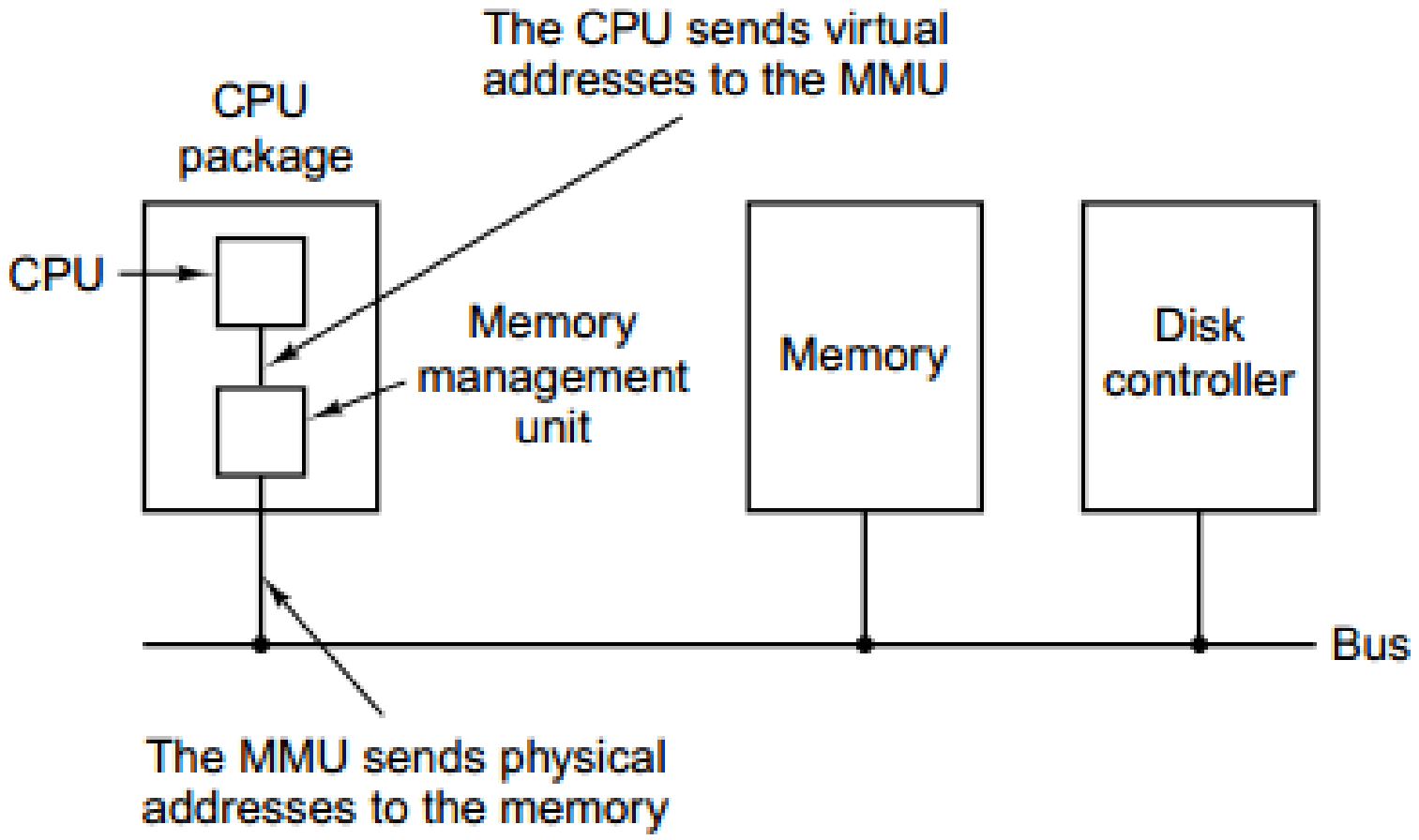
# Adresse logique/physique : Conversion

- La multiprogrammation et l'allocation dynamique ont engendré le besoin de lire un programme dans des positions différentes.
- Les adresses logiques sont traduites en adresses physiques après le chargement du programme en mémoire centrale par l'unité de traduction d'adresses, appelée aussi unité de gestion de la mémoire (MMU : Memory Management Unit)

# MMU : Unité de gestion de la mémoire

- L'unité de gestion de la mémoire (MMU) est une unité physique chargée par la conversion des adresses logiques en adresses physiques.
  - Au moment du chargement vers la mémoire physique, le **registre base** (appelé aussi registre de réallocation) est ajouté à toute adresse fournie par le processeur.

# MMU : Unité de gestion de la mémoire

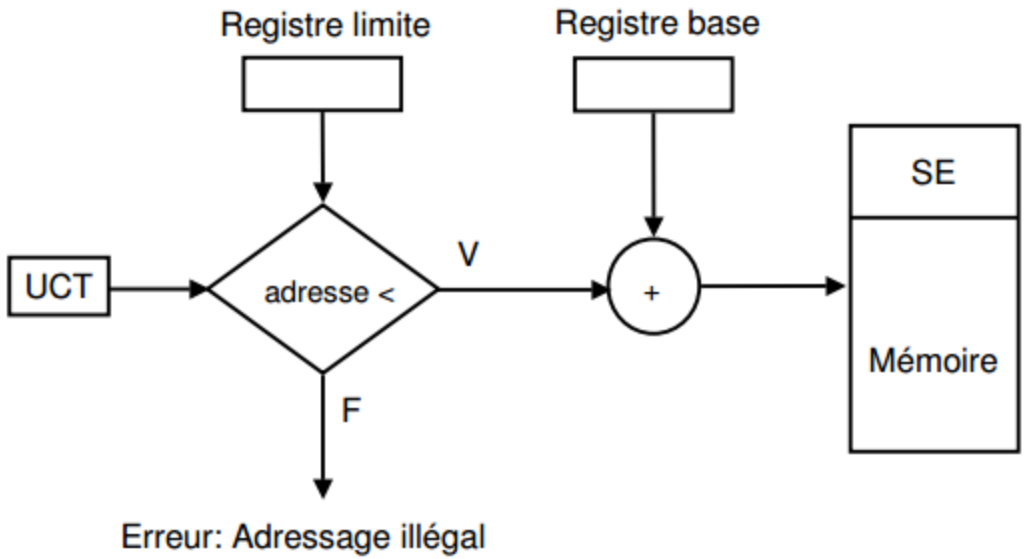


# Registres base et limite

- Pour protéger un processus d'être envahi par un autre, une solution consiste à utiliser les deux registres base et limite.
- Le registre **base** est utilisé pour stocker l'adresse de début (la limite inférieure) de la partition allouée à un processus et dans le registre **limite** la longueur de la partition (sa taille) ou l'adresse limite de la partition .
- Toute adresse générée par le processeur doit être ajoutée au registre **base** pour obtenir l'adresse physique.
- Le registre base est une notion plus utile pour résoudre le problème de déplacement des processus en mémoire.
  - Si un processus est déplacé en mémoire, il suffit alors de modifier son registre base.

# Registres base et limite

- Le chargement de l'espace d'adressage d'un processus en mémoire principale est obtenu en ajoutant le contenu du registre base à chaque adresse mémoire référencée.
- Le registre limite permet de vérifier dès le début si l'accès est autorisé ou non ; on vérifie si les adresses ne dépassent pas le registre limite.



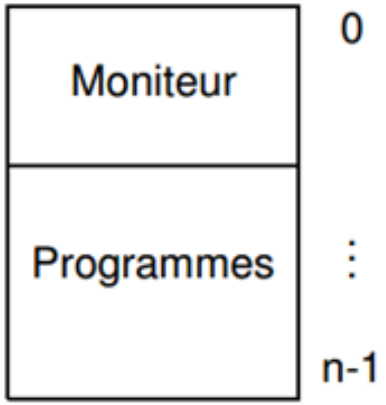
# Plan

- **Introduction**
- **Adresse physique/ Adresse logique**
- **Présentation de la mémoire dans un système**
  - Monoprogrammé (mono-processus)
  - Multiprogrammé (multiprocessus)
- **Techniques de gestion de la mémoire**

# Système monoprogrammé : Moniteur résident

- Dans ce système la mémoire est divisée en deux partitions :
  - une pour le **moniteur résident du système** (en général en mémoire basse);
    - Le moniteur résident englobe les principales composants d'un système d'exploitation.
  - et une autre pour les programmes

des utilisateurs (mémoire haute).



# Système monoprogrammé : Moniteur résident

- Le MS-DOS est un exemple typique de ce genre de système.
- Un seul processus utilisateur est en mémoire à un instant donné (cette technique n'est plus utilisée de nos jours).
  - L'utilisateur entre une commande sur un terminal et le système d'exploitation charge le programme demandé en mémoire puis l'exécute.
  - Lorsqu'il se termine, le système d'exploitation affiche une invitation sur le terminal et attend la commande suivante pour charger un nouveau processus qui remplace le précédent.

# Système monoprogrammé : Moniteur résident

- Avantages :
  - Simplicité
  - SE très réduit
- Inconvénients :
  - Mauvaise utilisation de la mémoire (un seul processus) et du processeur (attente des E/S).

# Plan

- **Introduction**
- **Adresse physique/ Adresse logique**
- **Présentation de la mémoire dans un système**
  - Monoprogrammé (mono-processus)
  - Multiprogrammé (multiprocessus)
- **Techniques de gestion de la mémoire**

# Système multiprogrammé

- La multiprogrammation permet l'exécution de plusieurs processus à la fois.
  - Lorsqu'un processus est bloqué en attente d'une E/S, un autre peut utiliser le processeur.
- Plus d'un processus en cours d'exécution à la fois signifie :
  - Maximiser le degré de la multiprogrammation.
  - Meilleure utilisation de la mémoire et du processeur.

# Système multiprogrammé

La multiprogrammation nécessite la présence de plusieurs processus en mémoire :

- La mémoire est donc partagée entre le système d'exploitation et plusieurs processus.
- Il se pose cependant le problème suivant :
  - Comment organiser la mémoire de manière à faire **cohabiter** efficacement plusieurs processus tout en assurant la **protection** des processus.

# Système multiprogrammé

Deux cas généraux à distinguer :

- **Multiprogrammation sans va-et-vient :** dans ce cas, un processus chargé en mémoire y séjournera jusqu'à ce qu'il se termine (pas de va-et-vient entre la mémoire et le disque).
- **Multiprogrammation avec va-et-vient :** dans ce cas, un processus peut être déplacé temporairement sur le disque (**mémoire de réserve : swap area ou backing store**) pour permettre le chargement et l'exécution d'autres processus.
  - Le processus déplacé sur le disque sera ultérieurement rechargé en mémoire pour lui permettre de poursuivre son exécution.

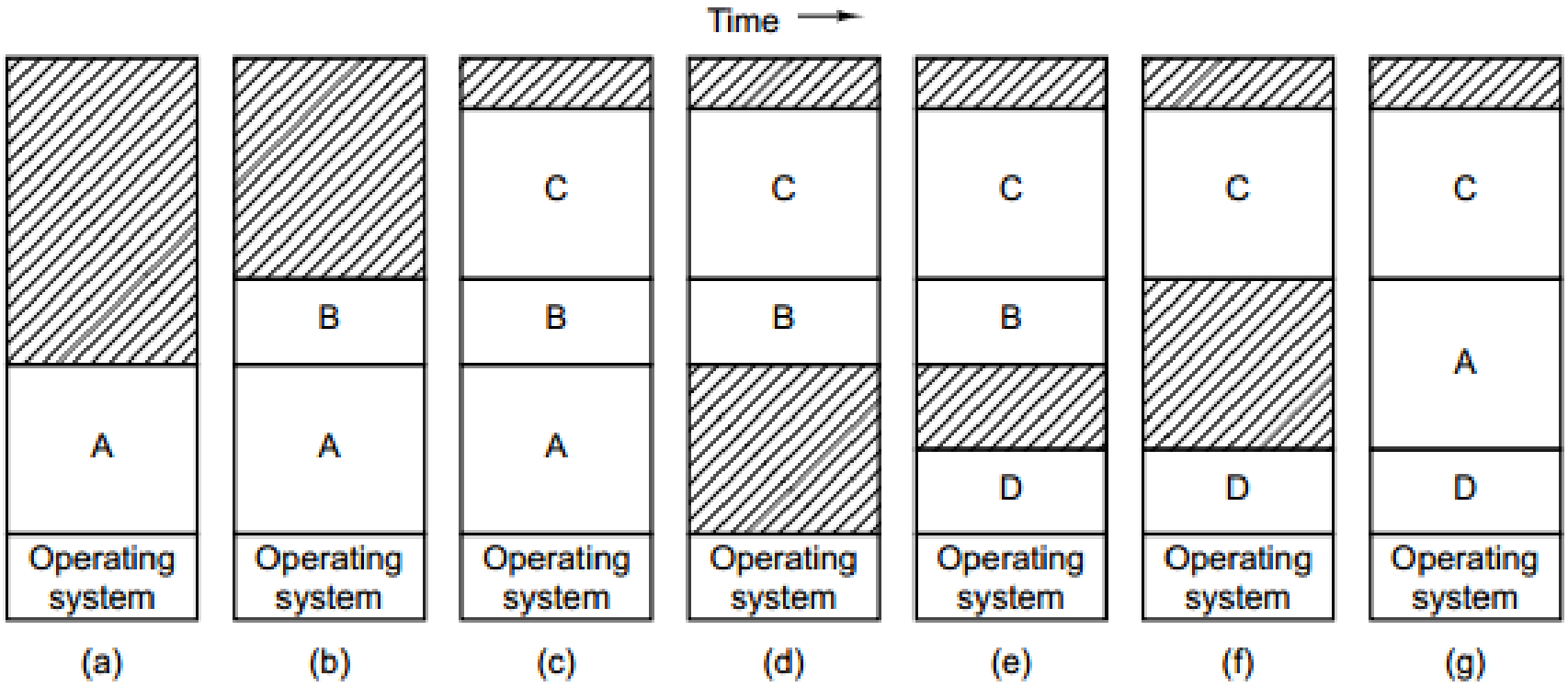
# Plan

- **Introduction**
- **Adresse physique/ Adresse logique**
- **Présentation de la mémoire dans un système**
- **Techniques de gestion de la mémoire**
  - **Swapping**
  - Partitionnement fixes sans va-et-vient
  - Partitionnement variables et va-et-vient

# Swapping

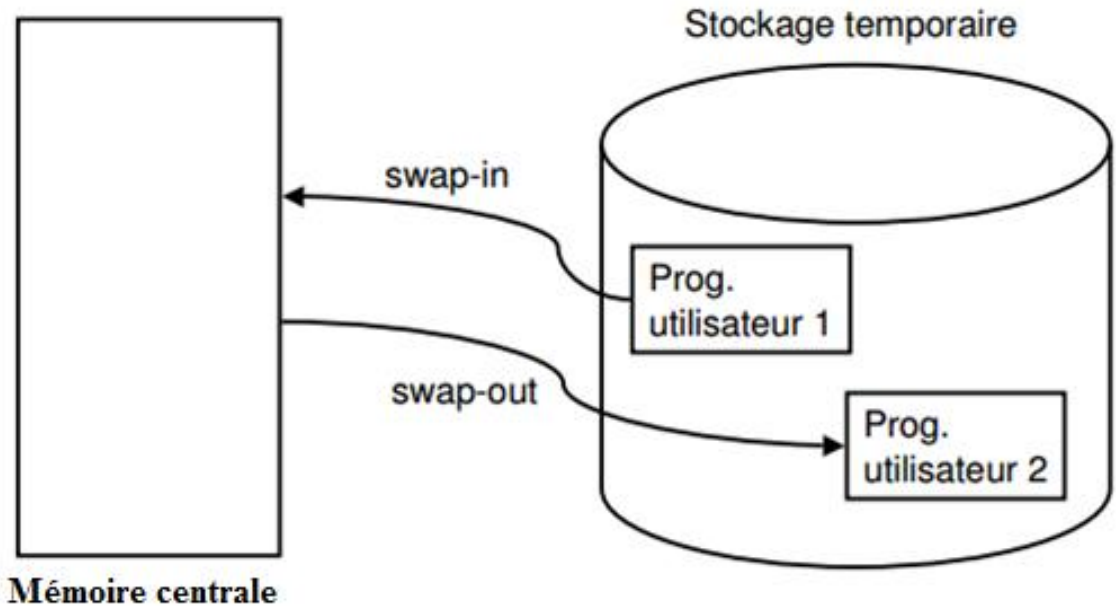
- Dans les systèmes multiprocessus, parfois la mémoire principale est insuffisante pour maintenir tous les processus actifs (résidents dans la file Prêts).
- Solution : Swapping
  - Déplacer temporairement certains processus sur une mémoire provisoire (en générale une partie réservée du disque, appelée *mémoire de réserve*, *swap area* ou *backing store*).
  - Le processus transféré sera ensuite ramené en mémoire pour continuer son exécution.

# Swapping : exemple



# Swapping

Le processus à transférer ne doit pas être en exécution.



# Swapping : Allocation

- Sur le disque, la zone de va-et-vient d'un processus peut être allouée à la demande dans la zone de va-et-vient générale.
- Quand un processus est déchargé de la mémoire centrale, on lui recherche une place.
- La zone de va-et-vient d'un processus peut aussi être allouée une fois pour toute au début de l'exécution.
- Lors du déplacement d'un processus de la mémoire centrale, on doit être sûr qu'il y a une zone d'attente libre sur le disque.

# Plan

- **Introduction**
- **Adresse physique/ Adresse logique**
- **Présentation de la mémoire dans un système**
- **Techniques de gestion de la mémoire**
  - Swapping
  - Partitionnement fixes sans va-et-vient
    - Fragmentation
    - File d'entrée
      - Allocation avec une seule file
      - Allocation avec plusieurs files
  - Partitionnement variables et va-et-vient

# Partitionnement Fixe
# Sans va-et-vient

Cette technique appelée aussi MFT (Mutiprogramming with a Fixed Number of Tasks),consiste à partitionner la mémoire en ***n* partitions de taille fixe**,

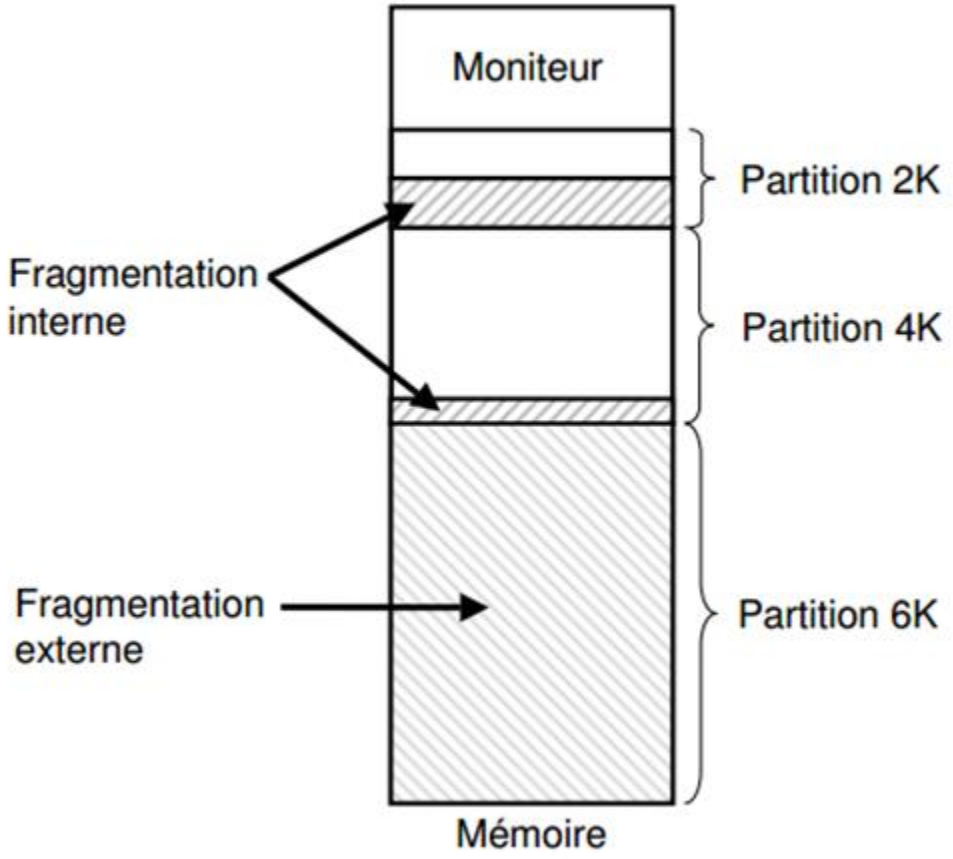
- Chaque partition peut contenir exactement 1 processus.
- Les partitions peuvent être de tailles égales ou inégales
- Le système d'exploitation doit maintenir une table indiquant les partitions libres et celles qui sont occupées.
- Le processus chargé résidera en mémoire jusqu'à ce qu'il se termine (pas de va-et-vient).

# Fragmentation

La gestion de mémoire selon le **MFT** pose le problème de **fragmentation :**

- L'espace non utilisé à l'intérieure d'une partition attribuée à un processus, est nommée **fragmentation interne**.
  - Il est très difficile que tous les processus aient exactement la même taille que celle de de la partition → Problème des **Trous.**
- Les partitions qui ne sont pas utilisées, donne le problème de la **fragmentation externe**.
  - Parfois il existe un espace mémoire total suffisant pour satisfaire une requête, mais il n'est pas contigu.

# Fragmentation

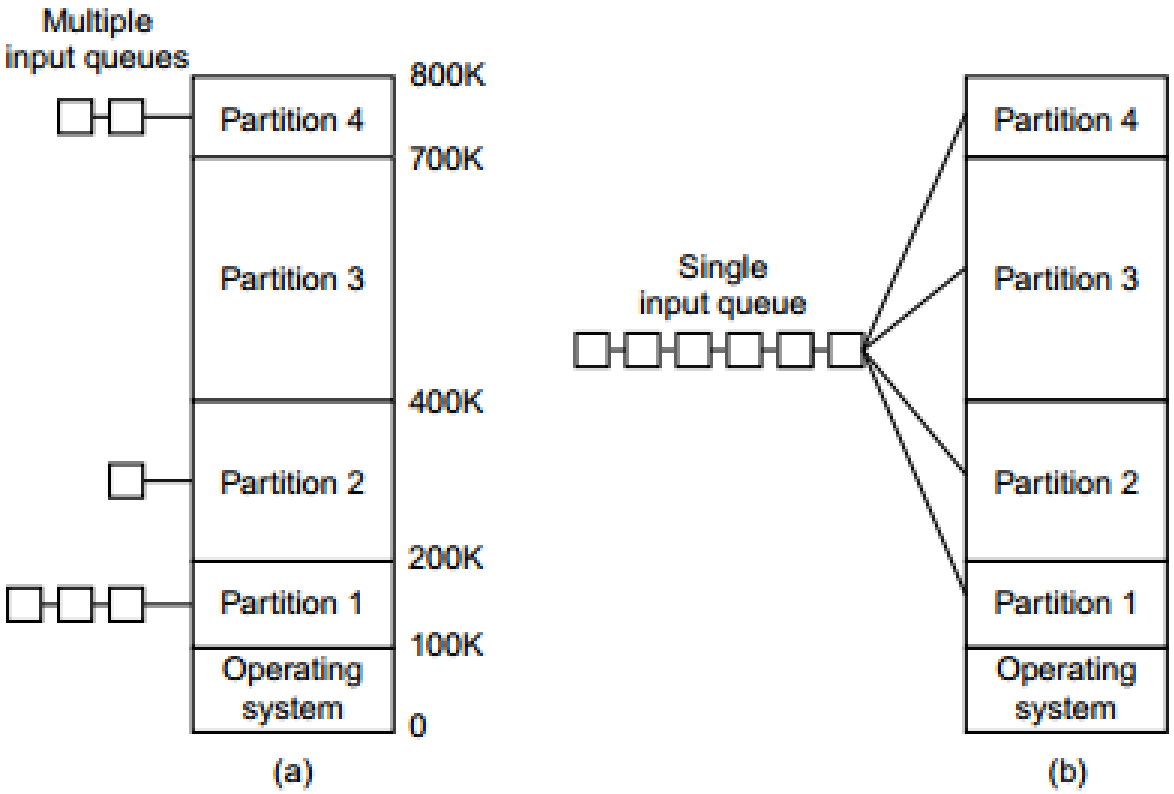


# Files d’entrée

Dans un environnement de partitionnement fixe, il y a deux façons pour allouer un espace mémoire à un processus :

- Files d’entrée multiples : une file pour chaque partition.
- File d’entrée unique : une seule file pour toutes les partitions.

# Files d'entrée



# Files d’entrée multiples

- Chaque nouveau processus est placé dans la file d’attente de la plus petite partition qui peut le contenir.
- L’espace inutilisé dans une partition allouée à un processus est perdu à cause de la fragmentation interne.
- Dans cette méthode il se peut que des processus reste en attente, le temps qu’il y a des partitions vides.
- La solution consiste à l’utilisation d’une seule file d’attente.

# File d’entrée unique

- Les processus sont placés dans une seule file d’attente jusqu’à ce qu’une partition se libère.
- Lorsqu’une partition se libère, il faut parcourir la file d’attente pour chercher le plus gros travail qui peut être placé dans la partition.
- Cet méthode pénalise les petits travaux.
- Solutions: Conserver au moins une **petite partition** afin que les petits travaux puissent s’exécuter.

# Plan

- **Introduction**
- **Adresse physique/ Adresse logique**
- **Présentation de la mémoire dans un système**
- **Techniques de gestion de la mémoire**
  - Swapping
  - Partitionnement fixes sans va-et-vient
  - Partitionnement variables avec va-et-vient
    - Compactage de mémoire
    - Allocation de mémoire

# Partitionnement variable avec va-et-vient

Dans cette méthode le nombre, la localisation et la taille des partitions varient dynamiquement (au cours du temps) selon les déplacements des processus.

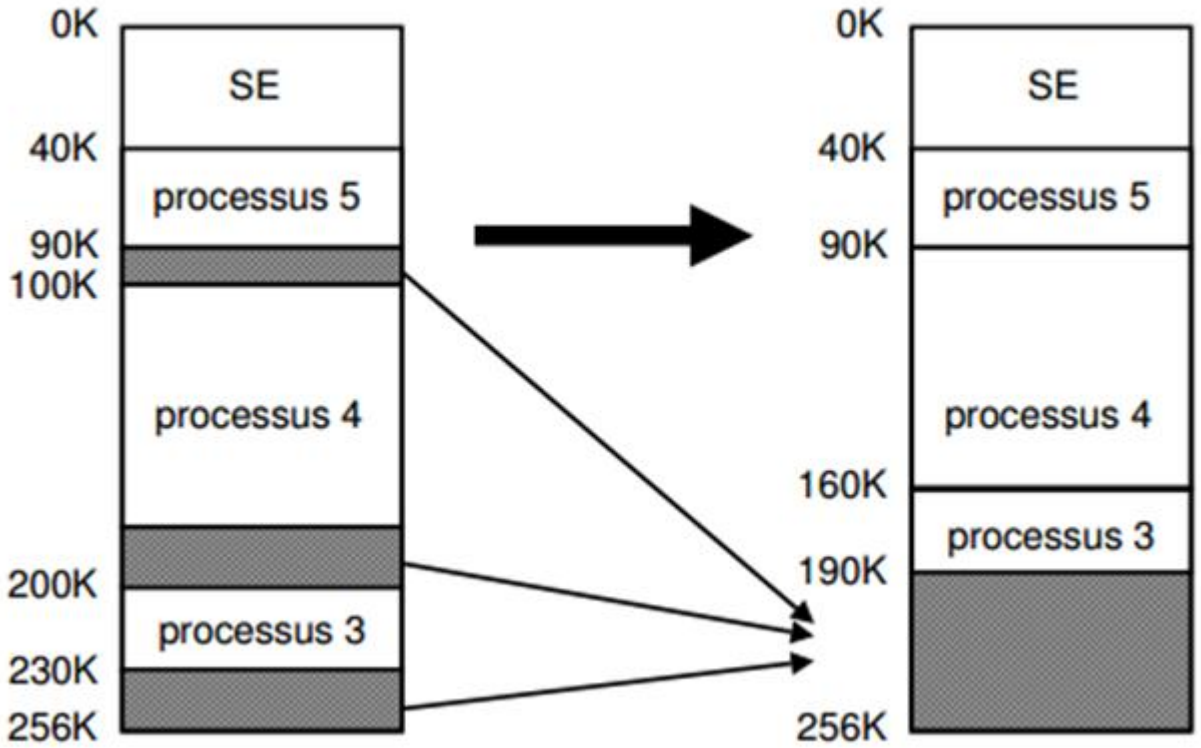
- Aussi appelée attribution de multiples partitions de mémoire à taille variable (**MVT**).
- Le système d'exploitation maintient une liste de toutes les partitions de mémoire non utilisées et de toutes celles utilisées.
  - ainsi quand un nouveau processus doit s'exécuter, on cherche un espace de mémoire assez convenable pour lui contenir.

# Compactage de mémoire

**Le compactage** est une solution proposée pour résoudre le problème de la fragmentation externe, dans un environnement de partitionnement variable :

- Il consiste à réunir les contenus de la mémoire afin de placer toute la mémoire libre ensemble dans un seule bloc.
- L'algorithme de compactage le plus simple consiste à déplacer tous les processus vers une extrémité de la mémoire; tous les trous se déplacent dans l'autre direction, produisant ainsi un grand trou de mémoire disponible

# Compactage de mémoire



# Partitionnement variable
# Avec va-et-vient : allocation de mémoire

- Processus de taille fixe :
  - L'allocation est simple dans ce cas, on cherche le bloc de mémoire le plus convenable à la taille du processus.
- Processus de taille variable :
  - Si les tailles des segments de données des processus peuvent varier, on suit les règles suivantes :
    - S'il existe un espace libre adjacent au processus, alors on lui donne l'accord de l'occuper.
    - Si cela n'est pas le cas, il faut soit le déplacer dans un emplacement plus grand qui peut le contenir, soit déplacer un ou plusieurs processus sur le disque pour créer un emplacement de taille suffisante.
    - Si la mémoire est saturée et si l'espace de va-et-vient sur le disque ne peut plus contenir de nouveaux processus, il faut tuer le processus qui a demandé de la mémoire.

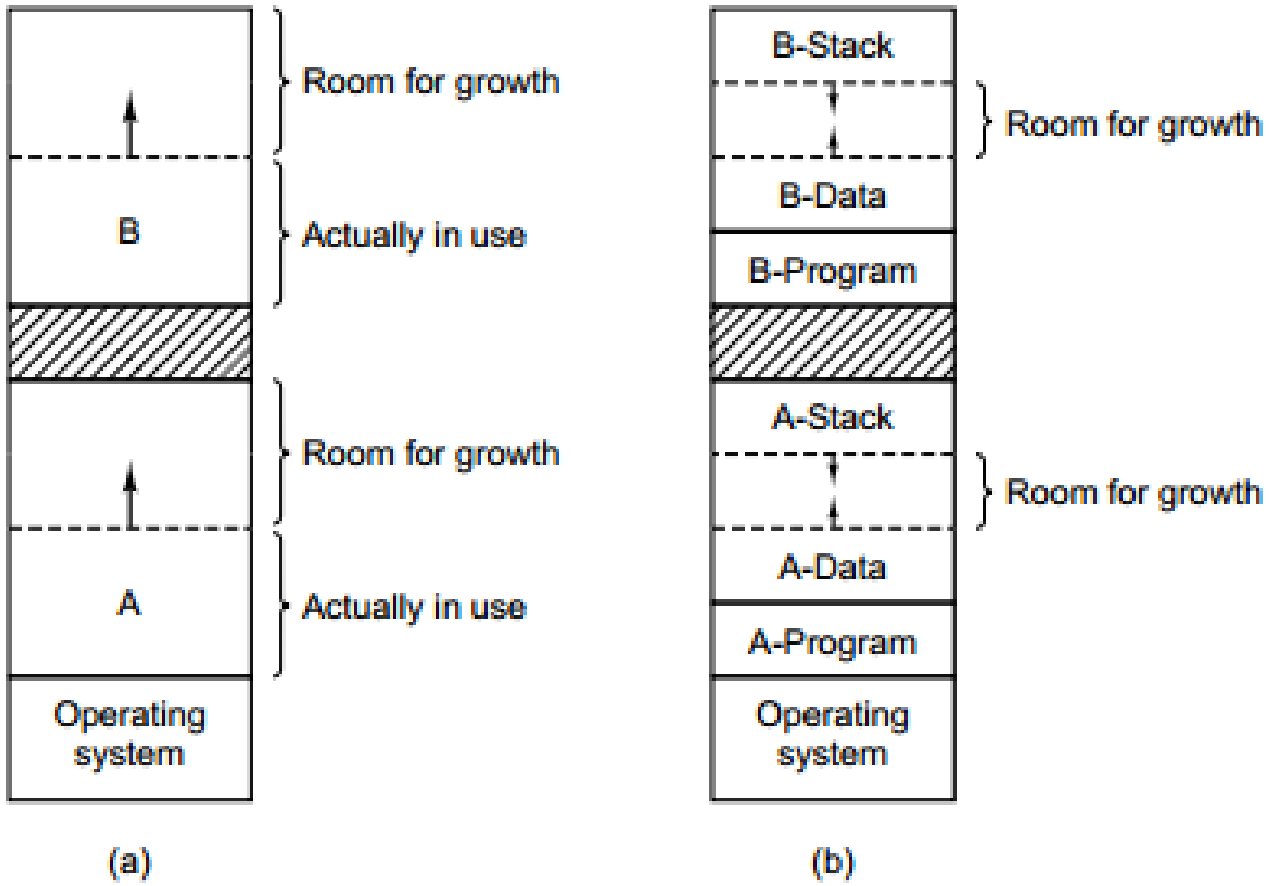
# Partitionnement variable
# Avec va-et-vient : allocation de mémoire

Processus de taille variable :

- Pour prévenir l'accroissement de la taille des processus au cours de leur exécution, on peut allouer à chaque processus chargé, un peu plus de mémoire qu'il en demande afin de réduire le temps système perdu pour cette gestion.

# Partitionnement variable
# Avec va-et-vient : allocation de mémoire

Processus de taille variable :



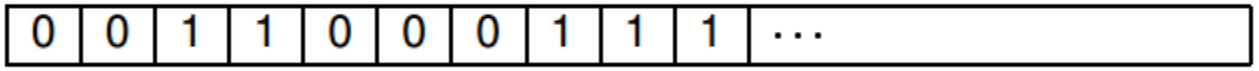
# Partitionnement variable
# Avec va-et-vient : allocation de mémoire

Pour gérer l'allocation et la libération de l'espace mémoire, le gestionnaire peut mémoriser l'état de la mémoire par :

- Les tables de bits ou bitmaps.
- Les listes chaînées.
- Les subdivisions.

# Gestion de la mémoire par table de bits

- La mémoire est divisée en unités d'allocation dont la taille peut varier de quelques mots à plusieurs Ko.
- A chaque unité, on fait correspondre un bit dans une **Table de bits (**bitmap) qui est mis à 0 si l'unité est libre et à 1 si elle est occupée.
- Plus l'unité d'allocation est faible plus la table de bits est importante.
  - En augmentant la taille de l'unité d'allocation, on réduit la taille de la table de bits mais on perd beaucoup de place mémoire (fragmentation interne).



# Gestion de la mémoire par table de bits

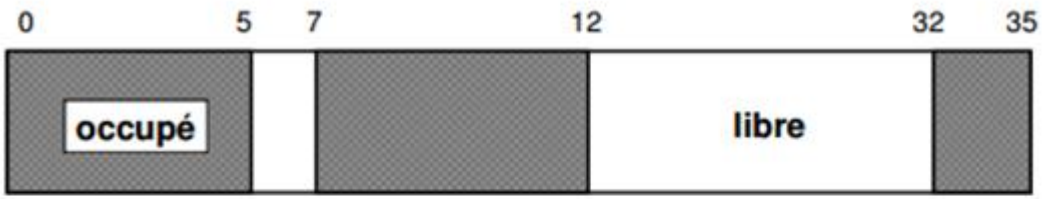
**Inconvénient :**

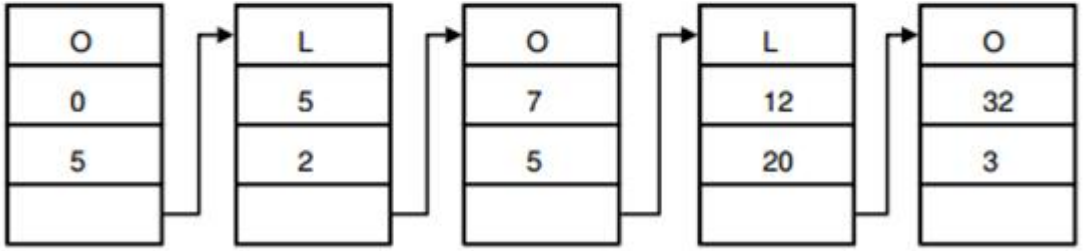
- Lorsqu'on doit ramener un processus de ***k*** unités, le gestionnaire de la mémoire doit alors parcourir la table de bits à la recherche de ***k*** zéros consécutifs.
- Cette technique est rarement utilisée car la méthode de recherche est lente.

# Gestion de la mémoire par liste chaînée

- Pour mémoriser l'occupation de la mémoire, on utilise une liste chaînée des segments libres et occupés.
  - Un segment est un ensemble **d'unités d'allocations** consécutives.
- Un élément de la liste est composé de quatre champs indiquant :
  - L'état du segment : libre ou occupé.
  - L'adresse de début du segment.
  - La longueur du segment.
  - Le pointeur sur l'élément suivant de la liste.
- Les listes peuvent être organisées en ordre :
  - de libération des zones,
  - croissant des tailles,
  - décroissant des tailles,
  - des adresses de zones.

# Gestion de la mémoire par liste chaînée



# Gestion de la mémoire par liste chaînée : Allocation de mémoire

L'attribution d'un espace libre à un processus peut se faire selon quatre stratégies principales :

- Le « premier ajustement » ou **first fit**
- Zone libre suivante
- Le « meilleur ajustement » ou **best fit**
- Plus grand résidu ou pire ajustement (**worst fit**)

# Gestion de la mémoire par liste chaînée : Allocation de mémoire

**Le « premier ajustement » :**

- Dans ce cas, on cherche le premier bloc libre de la liste qui peut contenir le processus qu'on désire charger.
- Inconvénient :
  - Fragmentation externe (Segments libres entre processus), si le processus n'occupe pas toute la zone.

# Gestion de la mémoire par liste chaînée : Allocation de mémoire

**Zone libre suivante :**

- Il est identique au précédent mais il mémorise en plus la position de l'espace libre trouvé.
- Ainsi, la recherche suivante commencera à partir de la dernière position trouvée et non à partir du début.

**Le « meilleur ajustement » ou best fit :**

- On cherche dans toute la liste la plus petite zone libre qui convient.
- On évite de fractionner une grande zone dont on pourrait avoir besoin ultérieurement.
- Inconvénient :
  - Cet algorithme est plus lent que le premier.
  - Il a tendance à remplir la mémoire avec de petites zones libres inutilisables.

# Gestion de la mémoire par liste chaînée : Allocation de mémoire

**Plus grand résidu ou pire ajustement (worst fit) :**

- Il consiste à prendre toujours la plus grande zone libre pour que la zone libre restante soit la plus grande possible.
- Les simulations ont montré que cette stratégie ne donne pas de bons résultats.

# Gestion de la mémoire par subdivision

Dans le schéma de **gestion par subdivision,** le gestionnaire de la mémoire mémorise une liste des blocs libres dont les tailles sont de 1, 2, 4, . . . 2^n octets jusqu'à la taille maximale de la mémoire. Initialement, on a un seul bloc libre.

**Bloc de taille *Si = 2^i octets***

- Lorsqu'il n'y a plus de bloc de taille *Si*, on subdivise un bloc de taille *Si+1* en deux blocs de taille *Si*.
- S'il n'y a plus de bloc de taille *Si+1* , on casse un bloc de taille *Si+2*, et ainsi de suite.
- Lors d'une libération de bloc de taille *Si*, si le bloc compagnon est libre, il y a lieu la reconstitution d'un bloc de taille *Si+1*.
- Pour un système binaire la relation de récurrence sur les tailles est donc :

***Si+1 = 2 * Si***